UNIVERSIDAD: Universidad Nacional de La Plata.

NUCLEO DISCIPLINARIO/COMITÉ ACADEMICO/OTROS TEMAS: Redes Académicas.

TITULO DEL TRABAJO: **RESOLUCION PARALELA DEL PROBLEMA PUZZLE N^2-1 SOBRE UN CLÚSTER.**

AUTOR(ES): Victoria María Sanz.

DIRECCIÓN: Ing. Armando E. De Giusti, Dr. Marcelo Naiouf, Lic. Franco Chichizola.

CORREOS ELECTRÓNICOS DE LOS AUTORES: vsanz@ciudad.com.ar.

PALABRAS CLAVES: Optimización, Algoritmos Paralelos, Cluster de computadoras.

Otimização, Algoritmos Paralelos, Clusters de computadores.

INTRODUCCIÓN

Los algoritmos de búsqueda en un espacio de estados pueden ser aplicados para resolver problemas de optimización discreta. El propósito de dichos algoritmos es encontrar una solución óptima que sea extremo de una función objetivo.

En la mayoría de los casos, este tipo de problemas tiene una gran demanda de procesamiento, ya que el espacio de búsqueda se vuelve exponencial, por lo que es imprescindible resolverlos en forma paralela.

Se investigó el problema del Puzzle N^2-1, generalización del problema propuesto por Sam Lloyd, y se estudió el algoritmo de búsqueda A*. Basado en dicho algoritmo, se presenta una solución secuencial al problema del puzzle y se realiza la paralelización sobre una arquitectura tipo cluster utilizando la librería MPI [1][2][3].

Problemas de optimización discreta (DOP)

Un problema de optimización discreta puede expresase como una tupla (S,f) donde:

- S es un conjunto finito o infinito contable de posibles soluciones.
- f es la función que aplica un costo a cada elemento de S (f: S => R).

Resolver este problema implicará encontrar un $x^* \in S$ tal que $f(x^*) \leq f(x)$, para todo $x \in S$.

En la mayoría de los problemas el conjunto S es bastante grande, y su enumeración exhaustiva para encontrar la solución óptima se torna imposible. Por este motivo, el problema se puede replantear como una búsqueda de un camino de costo mínimo en un grafo, que comienza desde la configuración inicial dada y finaliza en un nodo solución. Dicho grafo recibe el nombre de *espacio de estados*, y a cada nodo del grafo se lo llama *estado*.

Problema del Puzzle N^2-1

El problema del Puzzle N^2-1 es una generalización del Puzzle-15 ideado por Sam Lloyd [4]. Consiste en N^2-1 piezas numeradas de1 a N^2-1 colocadas en un tablero de tamaño N^2 [5].

N^2-1 casilleros del tablero contienen exactamente una pieza, quedando sólo una casilla vacía la cual se denomina “hueco”.

El objetivo del puzzle es repetidamente llenar el hueco con una pieza adyacente a él en sentido horizontal o vertical, hasta alcanzar un tablero donde en la casilla (i,j) se encuentra la pieza numerada como (i-1)*n+j y en la casilla (n,n) se encuentra el hueco.

La solución al problema planteado tendrá que ser aquella que minimice la cantidad de movimientos que deben realizarse para alcanzar la configuración final desde la configuración inicial dada.

La figura 1.a visualiza un Puzzle N^2-1 donde N es 4. La figura 1.b representa un esquema de solución al problema.

2	5	1	12
15	3	8	14
4	13	10	
11	6	9	7

Figura 1.a. Tablero inicial

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

Figura 1.b. Tablero final.

Construcción del espacio de estados del problema

Para construir el grafo sobre el cual se realizará la búsqueda se requiere:

- *Esquema de ramificación:* utilizado para generar subproblemas a partir de un problema dado.
- *Estrategia de búsqueda:* estrategia para seleccionar un nodo entre los nodos pendientes de acuerdo a prioridades definidas. Generalmente la estrategia se basa en seleccionar nodos que estén próximos a la solución.
- *Función de costo:* estima el costo de alcanzar la solución a partir de un nodo intermedio. Sea h(x) esta función (también llamada *heurística*), si h(x) es una cota inferior del costo real requerido para alcanzar el nodo solución a partir de x, entonces h(x) es admisible[1]. Sea g(x) la función que denota el costo de alcanzar el estado x a partir de la configuración inicial, luego la cota inferior para el camino que puede ser obtenido expandiendo el camino actual entre la configuración inicial y x es l(x) = g(x) + h(x).

Muchos algoritmos de búsqueda podrán determinar la solución óptima buscando sólo en una porción del grafo.

Distancia de Manhattan

Supongamos que cada posición del tablero se representa como un par ordenado. La distancia entre las posiciones (i,j) y (k,l) está definida como | i - k | + | j - l |. Dicha distancia recibe el nombre de *Distancia de Manhattan*.

[1] Función admisible h(x): el costo real para alcanzar el nodo objetivo a partir del nodo x siempre es mayor o igual al valor de h(x).

La suma de las distancias de Manhattan entre las posiciones del tablero x y el tablero final estimará el número de movimientos requeridos para transformar el tablero x en el tablero solución. Esta función se utilizará como función heurística admisible h(x).

En la figura 2 indica las distancias de Manhatan para algunos valores del tablero.

2	5	1	12
15	3	8	14
4	13	10	
11	6	9	7

Distancia de Manhattan (2) = 1
Distancia de Manhattan (15) = 4
Distancia de Manhattan (1) = 2
Distancia de Manhattan (12) = 2

Figura 2. Ejemplos de distancia Maniatan.

Algoritmo A*

A* es una de las variantes más conocidas de la búsqueda Best First Search.

Cada nodo n es valuado de acuerdo con el costo de alcanzar el mismo a partir de la raíz del árbol de búsqueda (g(n)) y una heurística que estima el costo para ir de n hasta un nodo solución (h(n)). Luego la función de costo será l(x) = g(x) + h(x).

Está garantizado que el algoritmo A* siempre encontrará la mejor solución.

Algoritmo A* secuencial

Este algoritmo mantiene una lista de nodos no explorados (lista abierta) ordenada de acuerdo al valor de la función l, y otra lista de nodos ya explorados (lista cerrada).

Inicialmente la lista abierta contiene un solo elemento, el nodo inicial, y la lista cerrada esta vacía.

En cada paso, el nodo con menor valor l (el *mejor nodo*) es removido de la lista abierta y es examinado. Si es el nodo solución, el algoritmo termina. Caso contrario, el nodo es expandido y es insertado en la lista cerrada. Cada nodo sucesor es insertado en la lista abierta sólo si no estaba en la lista cerrada, o estaba pero su valor l es menor al anterior.

La lista cerrada es utilizada para evitar trabajo repetido, de modo que no se expandirán nuevamente nodos ya explorados.

Dado que siempre se remueve el nodo con menor valor l, la lista abierta comúnmente se implementa con una cola de prioridades. La lista cerrada se puede implementar con una tabla de hash donde cada entrada contiene como clave un valor de la heurística h(x), y como valor una lista con todos los tableros cuyo h(x) es igual a la clave.

Si se requiere conocer el camino desde el nodo inicial hasta el nodo solución, cada nodo debe mantener una referencia a su nodo padre, o poseer la secuencia de pasos por la cual se llegó hasta él, así se podrá determinar cómo fue encontrado.

Algoritmo A* paralelo

La estrategia de paralelización consiste en mantener las listas *abierta* y *cerrada* locales para cada procesador.

Al principio sólo uno de los procesadores tendrá como trabajo el nodo inicial. A medida que son generados otros nodos, los procesadores recibirán los mismos para comenzar a trabajar. Todos los procesadores harán una búsqueda en el ámbito local, construyendo su propia lista cerrada, para evitar trabajo repetido localmente, como también su lista abierta local.

El árbol de búsqueda tiende a estar desbalanceado, por lo que una técnica de balance de carga (centralizada o distribuida) deberá ser implementada [6][7].

Los procesadores deberán comunicarse los valores mínimos de las soluciones encontradas, a fin de minimizar búsquedas innecesarias.

El criterio de terminación utilizado por el algoritmo secuencial falla, ya que en cada momento varios nodos de las listas abiertas están siendo expandidos. Es posible que al encontrar una solución no sea la mejor. Para solucionar este problema los procesadores deben seguir trabajando sobre nodos cuyo costo es menor que la mejor solución actual.

OBJETIVO

Investigar la superlinealidad en el speedup de los algoritmos paralelos de búsqueda en espacio de estados mediante el estudio del problema del Puzzle N.

DESARROLLO

Resolución del problema del Puzzle N^2-1. Solución Secuencial

En este trabajo se implementó la solución al problema Puzzle N^2-1 utilizando los conceptos de Distancia de Manhatan como función de heurística y el algoritmo A* como algoritmo base para la generación de la solución. A continuación se detallan los parámetros y el pseudocódigo del algoritmo.

Entrada: configuración inicial del juego (x) (notar que g(x) = 0).

Salida: camino mínimo que transforma la configuración inicial en la configuración final del juego.

```
Crear lista abierta.
Crear lista cerrada.
Insertar (lista abierta, x, h(x)).
mientras (lista abierta no vacía) y (no encontré solución)

    // Extraigo el mínimo nodo de la lista abierta, llamémoslo n
    n = EliminarMínimo (lista abierta)

    // Si el nodo era la solución, es decir h(n) = 0, termina el algoritmo.
    // Caso contrario se expande n.
    si (EsSolución(n))
       solución = n
    sino
       hijos = Expandir(n)
       Insertar(lista cerrada,n)

        // Un nodo es aceptable si no esta en la lista cerrada, o si está pero con un costo mayor al actual
       para cada uno de los hijos de n
              si (EsAceptable(hijo))
                     Insertar(lista abierta, hijo, h(hijo) + g(n) + 1)
Retornar solución.
```

Resolución del problema del Puzzle N^2-1. Solución Paralela

La solución paralela planteada no requiere un proceso central, sólo requiere procesos trabajadores que realicen la búsqueda en el espacio de estados del problema [8].

Para la implementación del algoritmo paralelo se utiliza la técnica de balance de carga distribuida “Asynchronous Round Robin” (ARR) y el algoritmo de “Terminación de Dijkstra” modificado.

Dados p procesos trabajadores, cada uno dispondrá de su lista abierta y lista cerrada de nodos, así como un valor indicando el costo de la mejor solución encontrada hasta el momento (CMS), que se utilizará para acotar la búsqueda.

El tablero inicial le es asignado al trabajador 0, proceso que también se encarga de detectar la terminación de la búsqueda.

Un proceso que posee trabajo en su lista abierta, a lo sumo procesa una cantidad fija de nodos en cada iteración (pasada como parámetro al algoritmo) o procesa nodos hasta encontrar una solución o que se vacíe su lista abierta.

A continuación el trabajador recibe, si los hay, costos de “mejores soluciones” encontradas por los demás y a medida que esto ocurre actualiza (si es necesario) su variable CMS. De esta manera los nodos a procesar serán sólo los que tengan costo menor a CMS.

Si el proceso todavía posee trabajo en su lista abierta, entonces examina si otros procesos le hicieron pedidos de trabajo, caso en el cual envía nodos del principio y final de su lista abierta al trabajador solicitante ocioso. Luego continúa trabajando sobre sus nodos. En caso contrario, el proceso está ocioso, por lo que envía un pedido de trabajo a su

donador siguiendo el algoritmo ARR. Si el proceso encontró una nueva solución, envía a todos los demás procesos el costo de la misma.

Luego espera los siguientes tipos de mensajes, que serán atendidos sin prioridad alguna:

- *Petición de trabajo:* un trabajador ocioso seleccionó a este proceso como su donador. Debido a que el proceso no dispone de trabajo, envía un mensaje de rechazo (según ARR).
- *Trabajo:* el donador envía el trabajo requerido. Ahora el proceso está activo nuevamente.
- *Rechazo de pedido de trabajo:* el donador seleccionado no tiene trabajo. El proceso debe enviar un mensaje de pedido de trabajo al próximo donador.
- *Token:* recepción del token para detección de terminación. En caso de ser necesario se actualiza el token y se pasa al siguiente proceso. El proceso 0, cada vez que recibe el token, chequea la terminación.
- *Nuevas soluciones encontradas por otros trabajadores:* en caso de ser necesario, se actualiza la variable CMS.

Cuando el proceso 0 detecta la terminación, envía un mensaje a los demás procesos avisando el fin del cómputo.

Se utilizó el token de terminación para trasladar los movimientos de la solución de costo mínimo hasta el proceso 0, de forma que los mensajes de comunicación de nuevas soluciones encontradas durante el algoritmo sólo posean un valor entero, el costo, evitando así overhead de comunicación.

RESULTADOS

Para las pruebas se dispone de un clúster homogéneo compuesto por 20 procesadores Pentium 4 de 2.4GHz y 1GB de memoria RAM, y 4 de ellos fueron utilizados para realizarlas [9].

Para las pruebas se consideraron diferentes tableros iniciales que requerían una cantidad variada de pasos para su solución. Además se consideraron tableros de 4*4, 5*5 y 6*6.

A continuación se muestran algunos resultados de las pruebas realizadas (para tableros de 4*4 y 6*6). Estos resultados se visualizan en tablas, las cuales incluyen entre

otros valores tales como desorden inicial dado por la Distancia de Manhatan; cantidad de nodos procesados, expandidos y podados en la solución; y por último el tiempo requerido por la solución expresado en segundos.

Profundidad solución	Desorden Inicial	Cant. Nodos procesados	Cant. nodos expandidos	Cant. nodos podados	Long.promedio de lista	Tiempo
30	18	19879↓	60888↓	30011↑	252↓	0,576872↓
32	26	8313↑	25641↑	15249↑	961↑	0,066417↑
30	18	25326	77205	27926	1055	15,768273
34	20	78269↑	242920↑	145391↑	820↓	7,21037↓
32	26	1203	3706	1241	41	0,056888
36	28	44873↑	136712↑	81213↑	428↓	1,644819↓
34	20	33656	104126	36852	1246	25,143015
38	26	92603↑	283592↑	146912↑	875↓	13,40615↓
36	28	40045	122496	43772	1251	31,386889
40	24	525124↑	1607612↑	959191↑	4637↓	459,425523↓
38	26	69068	215181	78157	2257	139,636074
42	30	1394372↑	4212014↑	2495914↑	10843↑	3044,37681↑
40	24	332260	1015948	382370	10383	3005,047988
42	30	324885	987051	357257	9024	2769,302766

Tabla 2. Programa Paralelo - Tablero 4*4

Tabla 1. Programa Secuencial - Tablero 4*4

Profundidad solución	Desorden Inicial	Cant. Nodos procesados	Cant. nodos expandidos	Cant. nodos podados	Long.promedio de lista	Tiempo
30	24	1589	5386	1633	58	0,050833
32	24	3621	12207	3728	129	0,152115
34	24	8122	28377	8361	280	1,140793
36	26	10594	36852	10982	341	2,543416
40	28	125646	435358	133522	3695	655,099622
42	30	54754	191094	57419	1564	97,517032
44	32	94969	328000	100562	2499	264,946125

Tabla 3. Programa Secuencial - Tablero 6*6

Profundidad solución	Desorden Inicial	Total nodos procesados	Total nodos expandidos	Total nodos podados	Long.promedio máx. de lista	Tiempo
30	24	2052↑	7028↑	4817↑	21↓	0,050533↓
32	24	4755↑	16076↑	10243↑	60↓	0,06479↓
34	24	9017↑	31697↑	16481↑	106↓	0,163491↓
36	26	4709↓	16557↓	10652↓	55↓	0,084008↓
40	28	121714↓	423090↓	257862↑	1000↓	34,176281↓
42	30	186661↑	652804↑	399966↑	1548↓	66,838854↓
44	32	95476↑	323875↓	183109↑	703↓	12,686173↓

Tabla 4. Programa Paralelo - Tablero 6*6

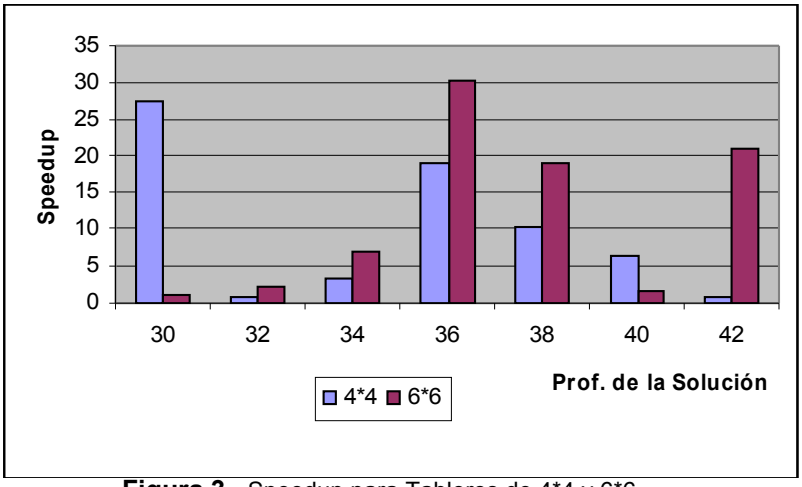

Figura 3. Speedup para Tableros de 4*4 y 6*6

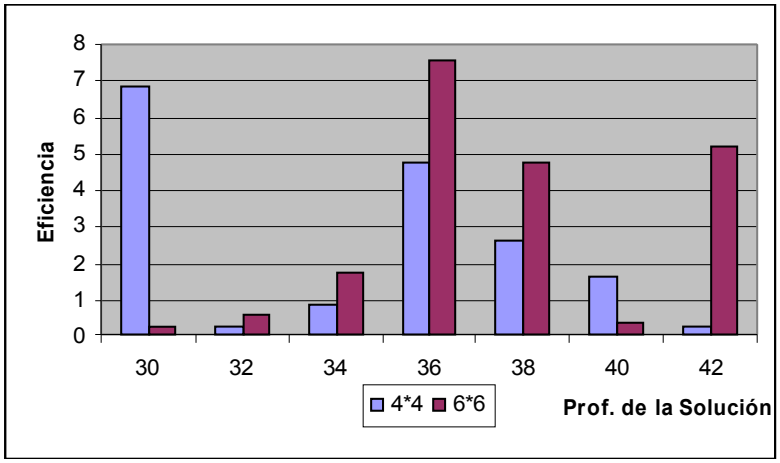

Figura 4. Eficiencia para Tableros de 4*4 y 6*6

La Figura 3 visualiza el speedup obtenido tanto en tableros de 4*4 como tableros de tamaños 6*6. En la misma se puede notar que para algunas pruebas se pudo obtener un speedup super lineal. La Figura 4 muestra la eficiencia obtenida para las mismas pruebas [10].

CONCLUSIONES Y LÍNEAS DE TRABAJO

En este trabajo se ha investigado la superlinealidad en el speedup de los algoritmos paralelos ejecutados sobre arquitecturas de cluster de PCs, para el caso de búsqueda en espacio de estados mediante el estudio del problema del Puzzle N^2-1.

Para esto se aplicó el algoritmo de búsqueda A* para su resolución secuencial y paralela. La implementación del algoritmo paralelo se realizó sobre una arquitectura tipo cluster utilizando la librería de pasaje de mensajes MPI.

Se obtuvieron resultados experimentales, en un cluster homogéneo y se analizó speedup y eficiencia.

Actualmente se trabaja en el análisis de problemas de orden N creciente, incrementando el número de máquinas en el cluster para relacionar la escalabilidad con la eficiencia.

En el futuro se tratará de evaluar soluciones similares sobre arquitecturas de multicluster heterogéneos y GRID de modo de caracterizar el overhead de comunicaciones que limita el máximo speedup alcanzable.

BIBLIOGRAFÍA

1. T. Anderson, D. Culler, D. Patterson, NOW Team, “A Case for NOW (Networks of Workstations)”, IEEE Micro, 15(1), 1995, pp. 54-64.
2. Snir M., Otto S., Huss-Lederman S., Walker D., Dongarra J., “MPI: The Complete Reference”, The MIT Press, Cambridge, Massachusetts ,1996.
3. Plamenka Borovska. “Parallel Combinatorial Search on Computer Cluster: Sam Loyd’s Puzzle”. International Conference on Computer Systems and Technologies - CompSysTech’06
4. M. Gardner. “The Mathematical Puzzles of Sam Loyd”. Dover, 1959.
5. D. Ratner and M. K. Warmuth. “The (n2 − 1)-puzzle and related relocation problems”. Journal for Symbolic Computation, 10:11–137, 1990.
6. A. Grama, A. Gupta, G. Karypis, V. Kumar. "Introduction to Parallel Computing", Pearson Addison Wesley, 2nd Edition, 2003.
7. M. J. Quiin.”Parallel Computing: Theory and Practice”. McGraw-Hill Companies; 2 Sub edition (September 1, 1993).
8. Hart Lambur, Blake Shaw. “Parallel State Space Searching Algorithms”. May 2004.
9. J. Basney, M. Livny. "Deploying a High Throughput Computing Cluster". R. Buyya Ed., High Performance Cluster Computing: Architectures and Systems, Vol. 1, Prentice-Hall, Upper Saddle River, NJ, USA, pp. 116-134, 1999.
10. Leopold C. "Parallel and distributed computing. A survey of models, paradigms, and approaches". Wiley Series on Parallel and Distributed Computing. Albert Zomaya Series Editor, 2001.